CREATION WEB DYNAMIQUE

III °) Programmation php

III-1 °) Introduction

Le PHP est un langage de script qui permet de dynamiser les sites web et générer des pages html selon des informations généralement stockées dans des bases de données.
Les langages de script comme le PHP sont exécutés à la demande par un programme chargé de l'interpréter. L'internaute demande l'exécution d'un script lorsqu'il consulte une page Web par l'intermédiaire de son navigateur Web. La demande est reçue par le serveur Web (par exemple Apache HTTPD), qui se rend compte qu'il doit la sous traiter à PHP. Ainsi un bon environnement pour faire une page dynamique nécessite : un serveur, un langage de script et un 'support' de données (de préférences une base de données).
Pour la suite de ce cours, nous utiliserons l'un des outils pratiques suivant : EasyPHP (Windows, Apache, MySQL, PHP) ou XAMPP (Windows ou Linux, Apache, MySQL, PHP et PERL).

III-2 °) Codage de base et mise en route

Basique ment, il faut connaitre les balises d'ouverture du script « <?php » et de fin « ?> », le mot clef « echo ou print» pour l'affichage, le « ; » pour terminer une ligne de commande , les accolades « {} » pour un bloc d'instruction, le signe « $ » pour les variables et ajouter à cela votre connaissance en programmation pour faire votre premier pas en PHP.

Exemple ;

```
<?php
echo "Bonjour l'invité";
?>
```

La lecture des données ce fait par url (($_GET['nomVariable ']) par un formulaire ($_POST['nomElementFormulaire']) et à travers des variables du programme ou des constantes definits.

Exemple : enregistrez le code ci-dessous dans un fichier nommé pagetest.php et lancer le dans votre navigateur.

```
<?php
if(empty($_GET['user']))
{
echo "Bonjour l'invité";
}
else
{
echo "Bonjour Mr ".$_GET['user'];
}
?>
```

Ajouter ceci « ?user=abdoul » dans votre url à la fin de localhost/pagetest.php

III-3 °) Syntaxe du langage php

Parenthèses

```
echo 1+2*3; //affiche "7"
```

```
echo (1+2)*3; //affiche "9"
```

Commentaires

```
//Commentaire sur une ligne
/*
Commentaire sur plusieurs lignes
Oui sur plusieurs lignes
*/
```

III-3-1 °) Les operateurs et types

Operateurs arithmétiques

```
echo 1-1; //soustraction
echo 1*1; //multiplication
echo 1/1; //division
echo 1%1; //modulo (reste de la division)
```

Operateurs d'affectation des variables

Variables numériques

```
$x = 5;
$x += 1; //ajoute à la valeur existante
$x -= 2; //soustrait à la valeur existante
$x *= 3; //multiplie la valeur existante
$x /= 4; //divise la valeur existante
echo $x; //affiche "3"
```

Variables chaines de caractères ;

```
$str = 'texte';
$str .= ' additionnel'; //concatène à la suite de la chaîne existante
echo $str; //affiche "texte additionnel"
```

Operateurs de comparaison

L'opérateur "=="

```
echo 1==1; //affiche "1"
echo 1==2; //n'affiche rien puisque c'est faux
echo 1==1.0; //affiche "1" puisque ce sont deux valeurs entières équivalentes
echo '1'==1.0; //affiche "1" puisque la chaîne "1" est équivalente à l'entier numérique "1"
echo '1'=='1.0'; //affiche "1" puisque leur valeur numérique évalue à la même valeur entière
```

L'opérateur "**==**" est un opérateur de comparaison de valeurs. Il ne tient pas compte du type de la valeur, puisquePHP est un langage à typage faible. L'opérateur "**!=**" est l'inverse de "**==**".

L'opérateur "==="

```
echo 1===2; //n'affiche rien puisque c'est faux
echo 1===1.0; //n'affiche rien puisque le type diffère (int et float)
echo '1'===1.0; //n'affiche rien puisque le type diffère (string et float)
echo '1'==='1.0'; //n'affiche rien puisque les deux chaînes ne sont pas égales
```

L'opérateur "**===**" est le même que le précédent, sauf qu'il n'effectue pas de conversion de type. Il est donc plus rapide que l'opérateur "**==**" et il ne donne pas les mêmes résultats. L'opérateur "**!==**" est l'inverse de "**===**".

Les opérateurs "**<>**" et "**!=**" sont identiques, ils permettent à des développeurs issus de différents horizons de s'adapter sans problème à PHP. Ils fonctionnent aussi bien pour les nombres que pour les chaînes.
Les opérateurs "**<**", "**>**", "**<=**" et "**>=**" sont assez classiques pour ne pas nécessiter d'explications.

Operateur conditionnel

L'opérateur ternaire "?" est une alternative : "est-ce vrai **?** *valeur si oui* **:** *valeur sinon*".

```
echo 1==1 ? 'vrai' : 'faux'; //affiche "vrai"
echo 1==2 ? 'vrai' : 'faux'; //affiche "faux"
echo (1==1) ? 'vrai' : 'faux';
echo (1==2) ? 'vrai' : 'faux';
```

Exemple :

```
//la valeur entière de $_GET['id'] ou zéro si la variable n'existe pas
$id = isset($_GET['id']) ? (int)$_GET['id'] : 0;
```

Operateurs Incrémentation / diminution

```php
$x = 5;
echo ++$x; //incrémente puis affiche 6
echo $x; //affiche 6
$x = 5;
echo $x++; //affiche 5 puis incrémente
echo $x; //affiche 6
$x = 5;
echo --$x; //réduit puis affiche 4
echo $x; //affiche 4
$x = 5;
echo $x--; //affiche 5 puis réduit
echo $x; //affiche 4
```

Operateurs comparaison logique

Comme dans tout langage de programmation, ces opérateurs permettent de vérifier plusieurs conditions à la fois dans un même test.

```php
if($variable > 2 and $variable < 10)
{
echo 'valeur entre 2 et 10 (exclus)';
}
if($variable > 2 && $variable < 10)
{
echo 'valeur entre 2 et 10 (exclus)';
}
if($variable > 2 or $variable < 10)
{
echo 'valeur supérieure à 2 ou inférieure à 10 (exclus)';
}
if($variable > 2 || $variable < 10)
{
echo 'valeur supérieure à 2 ou inférieure à 10 (exclus)';
}
```

• "**&&**" et "**and**" sont identiques sauf pour la priorité qui leur est attribuée ;
• "**||**" et "**or**" sont identiques sauf pour la priorité qui leur est attribuée.

```php
//différence entre "&&" et "and"
var_dump(0 and 0 || 1); // FALSE car équivalent à : 0 and (0 or 1)
var_dump(0 && 0 || 1); // TRUE car équivalent à : (0 and 0) or 1
//différence entre "||" et "or"
var_dump(1 or 0 and 0); // TRUE car équivalent à : 1 or (0 and 0)
var_dump(1 || 0 and 0); // FALSE car équivalent à : (1 or 0) and 0)
```

Les types PHP sont :

• **boolean** : un contraste "vrai" ou bien "faux", "blanc" ou "noir", "1" ou "0", "yin" ou "yang"... ;
• **integer** : une valeur numérique entière ;
• **double** : une valeur numérique flottante (à virgule) ;
• **string** : une chaîne de caractères (texte) ;
• **array** : un tableau (ensemble de valeurs) ;
• **object** : un objet (instance de classe) ;
• **resource** : une ressource (type abstrait, inutilisable par le programmeur, utilisé uniquement pour des fonctions) ;
• **NULL** : un type spécial qui désigne l'absence de valeur.

Fonction utiles pour les types : La fonction **var_dump()** affiche le type d'une variable et son contenu (ainsi que sa taille si c'est une chaîne). Elle s'applique aussi bien aux variables scalaires qu'aux objets, tableaux, ressources...

Le type String

Une chaîne de caractères peut s'écrire de diverses manières en PHP, chacune utilisant un "délimiteur" bien précis dont les plus utilisés sont :

```php
//Délimitation par des guillemets :
echo "Bonjour la classe!";
```

```
//Délimitation par des apostrophes :
echo 'Bonjour la classe';

echo "Voici un exemple d'apostrophe";
echo "Voici un exemple de \"guillemets\"";

echo 'Voici un exemple de "guillemets"';
echo 'Voici un exemple d\'apostrophe';

$user = 'Bouzou';
echo $user; //affiche "Bouzou"
echo $user[3]; //affiche "z" : gestion de la variable comme d'un tableau de caractères
echo $user{3}; //affiche "z" : gestion de la variable comme d'une chaîne (string)
echo substr($user, 3, 1); //affiche "z" : extraction d'un caractère à partir du 3ème
```

Le type numérique (int, float)

```
var_dump(1.2);
var_dump(1.0);
var_dump(1);

Affiche :
 float(1.2)
float(1)
int(1)
```

Les types spéciaux

• **NULL** : Valeur/variable vide ou inexistante ;
• **resource** : Par exemple une variable permettant d'identifier une connexion à une base de données ;
• **object** : Utilisé en Programmation Orientée Objet (POO, *cf.* plus loin).

Exemple : ressource fichier

```
$file = fopen(__FILE__, 'r');
echo $file;
var_dump($file);
fclose($file);

Affiche
Resource id #3
resource(3) of type (stream)
```

Fonctions utiles

En plus de **var_dump on retrouve :**

```
echo gettype($variable); //affiche le type de la variable
echo get_resource_type($variable); //affiche le type de la ressource
```

La fonction **settype()** permet de modifier le type d'une variable pendant l'exécution du programme. Des fonctions spécifiques **intval()**, **floatval()** et **strval()** permettent d'effectuer la même opération, et il est possible d'appliquer des "cast" à la mode du langage C.

```
$x = 5.2;
var_dump($x);
settype($x, 'int');
var_dump($x);
Affiche
float(5.2)
int(5)

var_dump(intval('5.2 ans'));
var_dump(floatval('5.2 ans'));
var_dump(strval('5.2 ans'));

int(5)
float(5.2)
string(7) "5.2 ans"
```

III-3-2 °) Les variables et constantes

Une variable PHP est identifiée par le signe dollar ($) suivi d'une lettre puis d'une suite de lettres, chiffres et traits soulignés (_).
Par convention, un nom de variable ne commence pas par une majuscule. S'il faut plusieurs mots pour composer le nom, ils sont habituellement séparés par des soulignés (_).

```
$variable; //ok
$variable_2; //ok
$Variable; //ok mais pas conventionnel (majuscule)
$2; //nom incorrect
$2 */-+$% = 'essai'; //nom incorrect
//La casse doit être respectée dans l'utilisation des variables
$var = 1;
$Var = 2;
// $var et $Var sont deux variables distinctes :
echo $var; //affiche "1"
echo $Var; //affiche "2"
```

Contrairement à d'autres langages, il n'est pas nécessaire de déclarer une variable avant de pouvoir lui affecter une valeur. Cependant, prenez garde à ne pas utiliser la valeur d'une variable avant d'être certain de lui avoir effectivement donné une valeur car cela lancerait un avertissement.

Les variables superglobales

Les variables superglobales sont mises en place par PHP lors du début du traitement d'une demande par Apache. Ces variables n'obéissent pas aux limites habituelles des variables en termes de visibilité à l'intérieur d'une fonction.
Elles sont accessibles de partout, c'est pourquoi elles portent le nom de "superglobales".

Voici les superglobales :

• $_GET : Les valeurs provenant de l'URL ;
• $_POST : Les valeurs envoyées par formulaire ;
• $_FILE : Les fichiers envoyés par formulaire ;
• $_SERVER : Les valeurs mises en place par le serveur Web (elles peuvent donc changer d'une configuration à l'autre) ;
• $_ENV : Les variables d'environnement (système d'exploitation) ;
• $_SESSION : Les valeurs mises dans le magasin des sessions ;
• $_COOKIE : Les valeurs transmises au moyen de cookies par le navigateur ;
• $GLOBALS : L'ensemble des variables du script.

Les constantes :

Une constante est un nom qui permet de donner une sémantique à une valeur. Cette valeur est figée pour toute la durée de l'exécution du script PHP. Par convention, on exclut les lettres minuscules dans le nom d'une constante.

Déclaration d'une constante

```
define('NOM_ADMIN', 'Bouzou'); //constante de type string
define('MAX_LIGNES', 5); //constante de type entier
echo NOM_ADMIN; //remarquez l'absence de guillemets et de signe dollar
```

Il existe des "constantes magiques" qui n'obéissent pas totalement aux règles des constantes. Elles existent dans tous les scripts sans qu'il soit nécessaire de les déclarer par programmation. On ne peut bien entendu pas les modifier par programmation, néanmoins leur valeur peut changer au fil de l'exécution du script.

• **__LINE__** : La ligne de code en cours ;
• **__FILE__** : Le nom complet du script en cours ;
• **__DIR__** : Le nom du répertoire du script en cours (depuis les versions 5.3 et 6.0 de PHP) ;

• **__FUNCTION__** : La fonction en cours ;
• **__CLASS__** : La classe en cours, similaire à **get_class($this)** ;
• **__METHOD__** : La méthode en cours ;
• **__NAMESPACE__** : L'espace de noms en cours (depuis les versions 5.3 et 6.0 de PHP).

AUTRES FONCTIONS UTILES :

Affichage

```
$x = 5;
echo $x; //affichage de la valeur
print $x; //affichage de la valeur
var_export($x); //affichage de la représentation PHP : utilisé lors des débogages du script
print_r($x); //affichage du contenu : utilisé lors des débogages du script
var_dump($x); //affichage du type et du contenu : utilisé lors des débogages du script
```

Test d'existence :

```
if(isset($variable))
{
//la variable existe
}
else
{
//la variable n'existe pas
}
```

Test de nullité :

```
if(empty($variable))
{
//la variable est nulle
}
else
{
//la variable est non nulle
}
```

Destruction

```
unset($variable);
```

III-3-3 °) Les tableaux

Un tableau est une variable contenant plusieurs valeurs. En PHP, les variables étant faiblement typées, les tableaux sont très simples à manipuler.

```
$nombres = array(3, 6, 9); //ce tableau contient trois valeurs
echo $nombres[1]; //accès direct à l'élément d'index 1, c'est-à-dire le deuxième élément
print_r($nombres); //affichage du tableau complet
```

Ajout à un tableau vide existant :

```
$nombres = array();
$nombres[] = 3; //ajout à la position suivante = zéro
$nombres[] = 6; //ajout à la position suivante = un
$nombres[] = 9; //ajout à la position suivante = deux
var_dump($nombres[1]); //accès direct à l'élément en position 1
print_r($nombres); //affichage complet
```

Autre exemple :

```
$tbstring = array('bouzou ', 'ndiaye ', 'touré'); //ce tableau contient trois tableaux de caractère.
echo $tbstring[1] [4]; //affiche y
```

Il existe plusieurs manières de parcourir un tableau :

```
$nombres = array(3, 6, 9);
```

```
foreach($nombres as $nombre)
{
echo $nombre.'<br/>';
}
Ou
foreach($nombres as $i => $nombre)
{
echo $i.' '.$nombre.'<br/>';
}

for($i=0; $i<count($nombres); ++$i)
{
echo $i.' '.$nombres[$i].'<br/>';
}

while(list($i, $nombre) = each($nombres))
{
echo $i.' '.$nombre.'<br/>';
}
```

Fonctions utiles pour les tableaux :

• **count()** : Compter le nombre d'éléments d'un tableau ;
• **sort()** : Trier un tableau (nombreuses fonctions disponibles) ;
• **array()** : *cf.* la documentation (nombreuses fonctions disponibles) ;
• **list()** : Assigne plusieurs valeurs en une opération (habituellement depuis un tableau) ;
• **current()** : Retourne l'élément du tableau désigné par son pointeur interne ;
• **reset()** : Réinitialise le pointeur interne du tableau ;
• **next()** : Avance le pointeur interne puis agit comme current() ;
• **prev()** : Recule le pointeur interne puis agit comme current().

III-3-4 °) Les structures de contrôle

Conditionnelle "if"

La structure conditionnelle if/else est l'une des plus classiques des langages de programmation. Elle permet d'effectuer des opérations ou d'autres opérations selon certaines conditions.

```
if(<expression>)
{
<instructions>
}
else
{
<instructions>
}
```

Alternative "switch"

```
switch(<expression>)
{
case <valeur 1>:
<instructions>
break;
case <valeur 2>:
<instructions>
break;
...
default:
<instructions>
}
```

Boucle "for"

```
for(<initialisation> ; <continuer tant que> ; <incrémentation>)
```

```
{
<instructions>
}
```

Boucle "while"

```
while(<continuer tant que>)
{
<instructions>
}
```

```
$file = fopen('file.ext', 'r'); //ouverture d'un fichier en lecture
while(!feof($file)) //tant que ce n'est pas la fin du fichier (End Of File)
{
echo fread($file, 8192); //lecture d'une ligne
}
fclose($file); //fermeture du descripteur de fichier
```

Une autre utilisation classique de la boucle "while" est de mettre une ligne d'affectation comme condition :

```
$db_result = mysql_query($sql);
while($row = mysql_fetch_assoc($db_result)) //tant qu'il y a un résultat
{
print_r($row);
}
```

Boucle "do while"

```
do
{
<instructions>
} while(<continuer tant que>);
```

Cette boucle fonctionne sur un principe similaire au "while", mais le premier tour de boucle est effectué avant toute chose.

Boucle "each"

Cette instruction n'est pas vraiment une boucle mais on l'utilise généralement conjointement à un type de boucle. Elle est prévue spécialement pour parcourir les tableaux :

```
$membres = array('Bouzou', 'ndiaye', 'Touré');
print_r(each($membres));
print_r(each($membres));
print_r(each($membres));
```

Afin de simplifier son utilisation, on la couple souvent avec une boucle "while" :

```
while(list($i, $membre) = each($membres))
{
echo $membre;
}
```

Boucle "foreach"

```
foreach(<tableau> as <element>)
{
<instructions>
}
```

```
foreach(<tableau> as <clef> => <element>)
{
<instructions>
}
```

Création de fonction en PHP

```
function <nom de la fonction> (<noms des paramètres>)
{
<instructions>
return <valeur>; //(optionnel)
}
```

Une fonction permet de réutiliser facilement du code PHP.
Une variable n'est visible que dans la fonction dans laquelle elle a été définie. Il existe cependant un mot clef "global" permettant d'outrepasser cette restriction : exemple : **global** $x;

III-4 °) La programmation orientée objet en php *(pour le niveau avancé)*

III-5 °) Configuration par le fichier php.ini

Le comportement de PHP est dicté par sa configuration, établie dans le fichier **php.ini**. Ce fichier standard de configuration est habituellement placé dans le même répertoire que PHP, et nous pouvons le modifier à l'aide de n'importe quel éditeur de texte.

Nous allons voir ici les directives les plus courantes du fichier *php.ini*, et la valeur recommandée dans chaque situation (soit que vous en **développement** ou **production)**

short_open_tag

Active ou désactive le tag "<?" (par opposition à "<?php") pour ouvrir un bloc PHP dans le script.

Configuration recommandée :
• Développement : Off ;
• Production : Off.

error_reporting

Définit quelles erreurs doivent être rapportées par PHP. Plus précisément les niveaux d'erreur. Un niveau élevé permet de connaître plus facilement les erreurs des scripts, donc les bugs ou encore les failles de sécurité. Il faut donc utiliser **E_ALL** | **E_STRICT** pour avoir toutes les erreurs.
error_reporting = E_ALL | E_STRICT

display_errors

Définit si PHP doit afficher les erreurs dans la sortie standard, indépendamment du niveau d'**error_reporting**.
Configuration recommandée :
• Développement : On ;
• Production : Off.

log_errors

Cette option devrait être laissée active dans toutes les situations. Sans fichier de log, vous n'auriez aucune idée des problèmes qui surviennent sur votre serveur, et vous n'auriez aucun moyen d'éviter leur récurrence.
Configuration recommandée :
• Développement : On ;
• Production : On.

register_globals

Elle permet d'importer les variables autoglobales dans la visibilité du script et pour des raisons de sécurité, il est conseillé de pas l'activer.
Configuration recommandée :
• Développement : Off ;
• Production : Off.

post_max_size

Taille maximum des données que PHP accepte depuis un formulaire POST. Donc à definir selon l'application.

upload_max_filesize

Taille maximum des fichiers que PHP accepte depuis un formulaire POST. Réduisez cette valeur au minimum, puis augmentez au niveau du répertoire (grâce au *httpd.conf* ou à un *.htaccess*) s'il y a besoin de plus.

mail function

Windows :

• SMTP (PHP_INI_ALL) : L'adresse du serveur SMTP à utiliser, typiquement votre hébergeur ou votre machine.
• smtp_port (PHP_INI_ALL) : 25 par défaut ;
• sendmail_from (facultatif - PHP_INI_ALL) : Adresse par défaut en tant qu'émetteur des e-mails.

Unix :

• sendmail_path (facultatif - PHP_INI_SYSTEM) : Chemin jusqu'au programme d'envoi d'e-mails, peut contenir des paramètres (par défaut : "sendmail -t -i").

III-6 °) Conclusion

Ce document représente que le support du cours, certaines éléments seront abordés en classe. Toutefois on y retrouve l'essentiel à savoir pour faire du php. Le prochain chapitre complétera le script PHP à savoir comment pour intégrer la gestion des données par une base de données.